Artificial Neural Networks: A Tutorial

Anil K. Jain
Michigan State University

Jianchang Mao
K.M. Mohiuddin
IBM Almaden Research Center

These massively parallel systems with large numbers of interconnected simple processors may solve a variety of challenging computational problems. This tutorial provides the background and the basics.

Numerous advances have been made in developing intelligent systems, some inspired by biological neural networks. Researchers from many scientific disciplines are designing artificial neural networks (ANNs) to solve a variety of problems in pattern recognition, prediction, optimization, associative memory, and control (see the "Challenging problems" sidebar).

Conventional approaches have been proposed for solving these problems. Although successful applications can be found in certain well-constrained environments, none is flexible enough to perform well outside its domain. ANNs provide exciting alternatives, and many applications could benefit from using them.[1-3]

This article is for those readers with little or no knowledge of ANNs to help them understand the other articles in this issue of *Computer*. We discuss the motivations behind the development of ANNs, describe the basic biological neuron and the artificial computational model, outline network architectures and learning processes, and present some of the most commonly used ANN models. We conclude with character recognition, a successful ANN application.

WHY ARTIFICIAL NEURAL NETWORKS?

The long course of evolution has given the human brain many desirable characteristics not present in von Neumann or modern parallel computers. These include

- massive parallelism,
- distributed representation and computation,
- learning ability,
- generalization ability,
- adaptivity,
- inherent contextual information processing,
- fault tolerance, and
- low energy consumption.

It is hoped that devices based on biological neural networks will possess some of these desirable characteristics.

Modern digital computers outperform humans in the domain of numeric computation and related symbol manipulation. However, humans can effortlessly solve complex perceptual problems (like recognizing a man in a crowd from a mere glimpse of his face) at such a high speed and extent as to dwarf the world's fastest computer. Why is there such a remarkable difference in their performance? The biological neural system architecture is completely different from the von Neumann architecture (see Table 1). This difference significantly affects the type of functions each computational model can best perform.

Numerous efforts to develop "intelligent" programs based on von Neumann's centralized architecture have not resulted in general-purpose intelligent programs. Inspired by biological neural networks, ANNs are massively parallel computing systems consisting of an exremely large number of simple processors with many interconnections. ANN models attempt to use some "organizational" principles believed to be used in the human

Challenging problems

Let us consider the following problems of interest to computer scientists and engineers.

Pattern classification

The task of pattern classification is to assign an input pattern (like a speech waveform or handwritten symbol) represented by a feature vector to one of many prespecified classes (see Figure A1). Well-known applications include character recognition, speech recognition, EEG waveform classification, blood cell classification, and printed circuit board inspection.

Clustering/categorization

In clustering, also known as unsupervised pattern classification, there are no training data with known class labels. A clustering algorithm explores the similarity between the patterns and places similar patterns in a cluster (see Figure A2). Well-known clustering applications include data mining, data compression, and exploratory data analysis.

Function approximation

Suppose a set of n labeled training patterns (input-output pairs), $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, have been generated from an unknown function $\mu(\mathbf{x})$ (subject to noise). The task of function approximation is to find an estimate, say $\hat{\mu}$, of the unknown function μ (Figure A3). Various engineering and scientific modeling problems require function approximation.

Prediction/forecasting

Given a set of n samples $\{y(t_1), y(t_2), \ldots, y(t_n)\}$ in a time sequence, $t_1, t_2, \ldots, t_n$, the task is to predict the sample $y(t_{n+1})$ at some future time t_{n+1}. Prediction/forecasting has a significant impact on decision-making in business, science, and engineering. Stock market prediction and weather forecasting are typical applications of prediction/forecasting techniques (see Figure A4).

Optimization

A wide variety of problems in mathematics, statistics, engineering, science, medicine, and economics can be posed as optimization problems. The goal of an optimization algorithm is to find a solution satisfying a set of constraints such that an objective function is maximized or minimized. The Traveling Salesman Problem (TSP), an *NP-complete* problem, is a classic example (see Figure A5).

Content-addressable memory

In the von Neumann model of computation, an entry in memory is accessed only through its address, which is independent of the content in the memory. Moreover, if a small error is made in calculating the address, a completely different item can be retrieved. Associative memory or content-addressable memory, as the name implies, can be accessed by their content. The content in the memory can be recalled even by a partial input or distorted content (see Figure A6). Associative memory is extremely desirable in building multimedia information databases.

Control

Consider a dynamic system defined by a tuple $\{u(t), y(t)\}$, where $u(t)$ is the control input and $y(t)$ is the resulting output of the system at time t. In model-reference adaptive control, the goal is to generate a control input $u(t)$ such that the system follows a desired trajectory determined by the reference model. An example is engine idle-speed control (Figure A7).

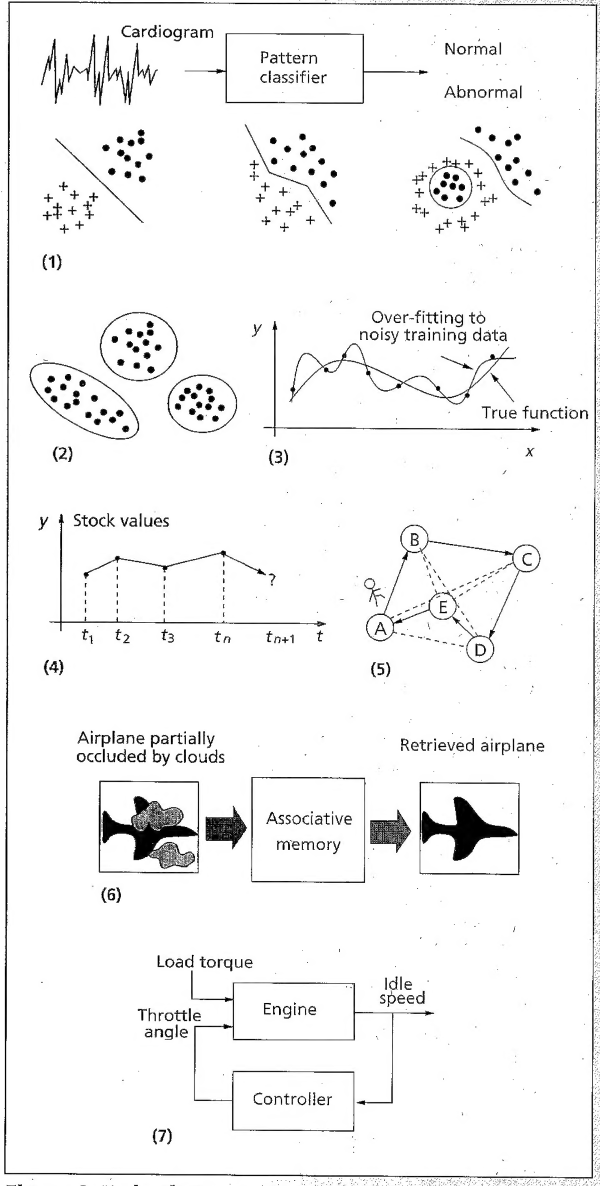


Figure A. Tasks that neural networks can perform: (1) pattern classification; (2) clustering/categorization; (3) function approximation; (4) prediction/forecasting; (5) optimization (a TSP problem example); (6) retrieval by content; and (7) control (engine idle speed). (Adapted from *DARPA Neural Network Study*[1])

brain. Modeling a biological nervous system using ANNs can also increase our understanding of biological functions. State-of-the-art computer hardware technology (such as VLSI and optical) has made this modeling feasible.

A thorough study of ANNs requires knowledge of neurophysiology, cognitive science/psychology, physics (statistical mechanics), control theory, computer science, artificial intelligence, statistics/mathematics, pattern recognition, computer vision, parallel processing, and hardware (digital/analog/VLSI/optical). New developments in these disciplines continuously nourish the field. On the other hand, ANNs also provide an impetus to these disciplines in the form of new tools and representations. This symbiosis is necessary for the vitality of neural network research. Communications among these disciplines ought to be encouraged.

Brief historical review

ANN research has experienced three periods of extensive activity. The first peak in the 1940s was due to McCulloch and Pitts' pioneering work.[4] The second occurred in the 1960s with Rosenblatt's perceptron convergence theorem[5] and Minsky and Papert's work showing the limitations of a simple perceptron.[6] Minsky and Papert's results dampened the enthusiasm of most researchers, especially those in the computer science community. The resulting lull in neural network research lasted almost 20 years. Since the early 1980s, ANNs have received considerable renewed interest. The major developments behind this resurgence include Hopfield's energy approach[7] in 1982 and the back-propagation learning algorithm for multilayer perceptrons (multilayer feedforward networks) first proposed by Werbos,[8] reinvented several times, and then popularized by Rumelhart et al.[9] in 1986. Anderson and Rosenfeld[10] provide a detailed historical account of ANN developments.

Biological neural networks

A *neuron* (or nerve cell) is a special biological cell that processes information (see Figure 1). It is composed of a cell body, or *soma*, and two types of out-reaching tree-like branches: the *axon* and the *dendrites*. The cell body has a nucleus that contains information about hereditary traits and a plasma that holds the molecular equipment for producing material needed by the neuron. A neuron receives signals (impulses) from other neurons through its dendrites (receivers) and transmits signals generated by its cell body along the axon (transmitter), which eventually branches into strands and substrands. At the terminals of these strands are the *synapses*. A synapse is an elementary structure and functional unit between two neurons (an axon strand of one neuron and a dendrite of another). When the impulse reaches the synapse's terminal, certain chemicals called neurotransmitters are released. The neurotransmitters diffuse across the synaptic gap, to enhance or inhibit, depending on the type of the synapse, the receptor neuron's own tendency to emit electrical impulses. The synapse's effectiveness can be adjusted by the signals passing through it so that the synapses can *learn* from the activities in which they participate. This dependence on history acts as a memory, which is possibly responsible for human memory.

The cerebral cortex in humans is a large flat sheet of neurons about 2 to 3 millimeters thick with a surface area of about 2,200 cm^2, about twice the area of a standard computer keyboard. The cerebral cortex contains about 10^{11} neurons, which is approximately the number of stars in the Milky Way.[11] Neurons are massively connected, much more complex and dense than telephone networks. Each neuron is connected to 10^3 to 10^4 other neurons. In total, the human brain contains approximately 10^{14} to 10^{15} interconnections.

Neurons communicate through a very short train of pulses, typically milliseconds in duration. The *message* is modulated on the pulse-transmission frequency. This frequency can vary from a few to several hundred hertz, which is a million times slower than the fastest switching speed in electronic circuits. However, complex perceptual decisions such as face recognition are typically made by humans within a few hundred milliseconds. These decisions are made by a network of neurons whose operational speed is only a few milliseconds. This implies that the computations cannot take more than about 100 serial stages. In other

Table 1. Von Neumann computer versus biological neural system.

	Von Neumann computer	Biological neural system
Processor	Complex High speed One or a few	Simple Low speed A large number
Memory	Separate from a processor Localized Noncontent addressable	Integrated into processor Distributed Content addressable
Computing	Centralized Sequential Stored programs	Distributed Parallel Self-learning
Reliability	Very vulnerable	Robust
Expertise	Numerical and symbolic manipulations	Perceptual problems
Operating environment	Well-defined, well-constrained	Poorly defined, unconstrained

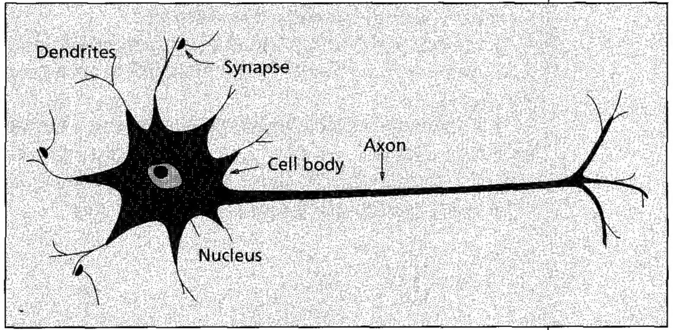


Figure 1. A sketch of a biological neuron.

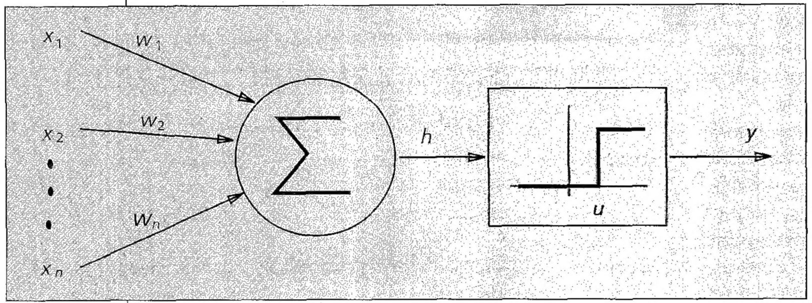


Figure 2. McCulloch-Pitts model of a neuron.

words, the brain runs parallel programs that are about 100 steps long for such perceptual tasks. This is known as the *hundred step rule*.[12] The same timing considerations show that the amount of information sent from one neuron to another must be very small (a few bits). This implies that critical information is not transmitted directly, but captured and distributed in the interconnections—hence the name, *connectionist* model, used to describe ANNs.

Interested readers can find more introductory and easily comprehensible material on biological neurons and neural networks in Brunak and Lautrup.[11]

ANN OVERVIEW

Computational models of neurons

McCulloch and Pitts[4] proposed a binary threshold unit as a computational model for an artificial neuron (see Figure 2).

This mathematical neuron computes a weighted sum of its n input signals, x_j, $j = 1, 2, \ldots, n$, and generates an output of 1 if this sum is above a certain threshold u. Otherwise, an output of 0 results. Mathematically,

$$y = \theta\left(\sum_{j=1}^{n} w_j x_j - u\right),$$

where $\theta(\cdot)$ is a unit step function at 0, and w_j is the synapse weight associated with the jth input. For simplicity of notation, we often consider the threshold u as another weight $w_0 = -u$ attached to the neuron with a constant input $x_0 = 1$. Positive weights correspond to *excitatory* synapses, while negative weights model *inhibitory* ones. McCulloch and Pitts proved that, in principle, suitably chosen weights let a synchronous arrangement of such neurons perform universal computations. There is a crude analogy here to a biological neuron: wires and interconnections model axons and dendrites, connection weights represent synapses, and the threshold function approximates the activity in a soma. The McCulloch and Pitts model, however, contains a number of simplifying assumptions that do not reflect the true behavior of biological neurons.

The McCulloch-Pitts neuron has been generalized in many ways. An obvious one is to use activation functions other than the threshold function, such as piecewise linear, sigmoid, or Gaussian, as shown in Figure 3. The sigmoid function is by far the most frequently used in ANNs. It is a strictly increasing function that exhibits smoothness and has the desired asymptotic properties. The standard sigmoid function is the *logistic* function, defined by

$$g(x) = 1/(1 + \exp\{-\beta x\}),$$

where β is the slope parameter.

Network architectures

ANNs can be viewed as weighted directed graphs in which artificial neurons are nodes and directed edges (with weights) are connections between neuron outputs and neuron inputs.

Based on the connection pattern (architecture), ANNs can be grouped into two categories (see Figure 4):

- *feed-forward* networks, in which graphs have no loops, and
- *recurrent* (or *feedback*) networks, in which loops occur because of feedback connections.

In the most common family of feed-forward networks, called multilayer perceptron, neurons are organized into layers that have unidirectional connections between them. Figure 4 also shows typical networks for each category.

Different connectivities yield different network behaviors. Generally speaking, feed-forward networks are *static*, that is, they produce only one set of output values rather than a sequence of values from a given input. Feed-forward networks are memory-less in the sense that their response to an input is independent of the previous network state. Recurrent, or feedback, networks, on the other hand, are dynamic systems. When a new input pattern is presented, the neuron outputs are computed. Because of the feedback paths, the inputs to each neuron are then modified, which leads the network to enter a new state.

Different network architectures require appropriate learning algorithms. The next section provides an overview of learning processes.

Learning

The ability to learn is a fundamental trait of intelligence. Although a precise definition of learning is difficult to formulate, a learning process in the ANN context can be viewed as the problem of updating network architecture and connection weights so that a network can efficiently perform a specific task. The network usually must learn the connection weights from available training patterns. Performance is improved over time by iteratively updating the weights in the network. ANNs' ability to automatically *learn from examples* makes them attractive and exciting. Instead of following a set of *rules* specified by human experts, ANNs appear to learn underlying rules (like input-output relationships) from the given collection of representative examples. This is one of the major advantages of neural networks over traditional expert systems.

To understand or design a learning process, you must first have a model of the environment in which a neural network operates, that is, you must know what information is available to the network. We refer to this model as

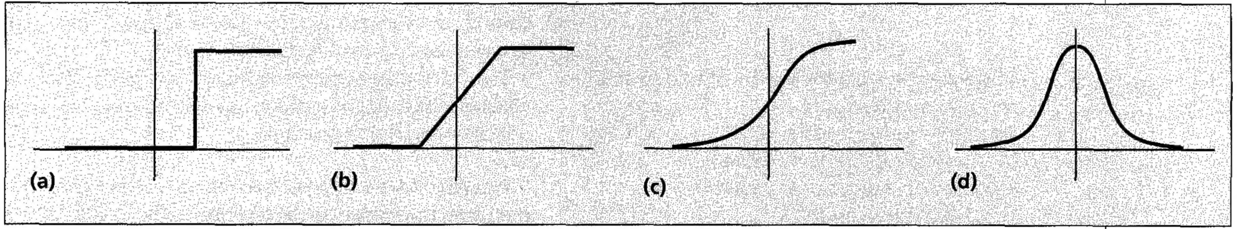


Figure 3. Different types of activation functions: (a) threshold, (b) piecewise linear, (c) sigmoid, and (d) Gaussian.

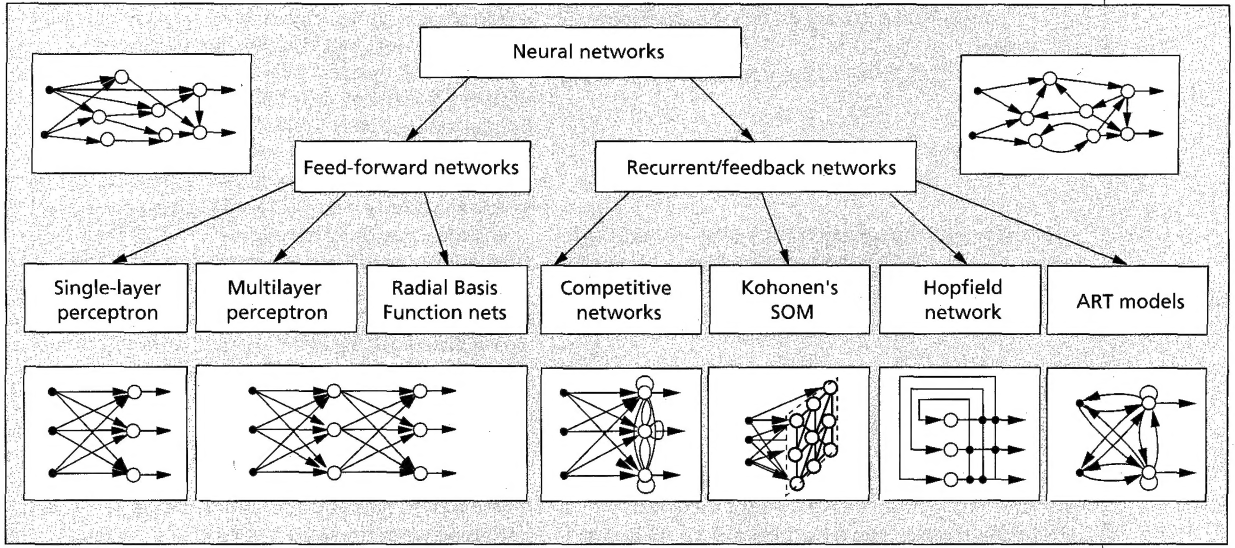


Figure 4. A taxonomy of feed-forward and recurrent/feedback network architectures.

a learning paradigm.[3] Second, you must understand how network weights are updated, that is, which *learning rules* govern the updating process. A *learning algorithm* refers to a procedure in which learning rules are used for adjusting the weights.

There are three main learning paradigms: supervised, unsupervised, and hybrid. In supervised learning, or learning with a "teacher," the network is provided with a correct answer (output) for every input pattern. Weights are determined to allow the network to produce answers as close as possible to the known correct answers. Reinforcement learning is a variant of supervised learning in which the network is provided with only a critique on the correctness of network outputs, not the correct answers themselves. In contrast, unsupervised learning, or learning without a teacher, does not require a correct answer associated with each input pattern in the training data set. It explores the underlying structure in the data, or correlations between patterns in the data, and organizes patterns into categories from these correlations. Hybrid learning combines supervised and unsupervised learning. Part of the weights are usually determined through supervised learning, while the others are obtained through unsupervised learning.

Learning theory must address three fundamental and practical issues associated with learning from samples: capacity, sample complexity, and computational complexity. Capacity concerns how many patterns can be stored, and what functions and decision boundaries a network can form.

Sample complexity determines the number of training patterns needed to train the network to guarantee a valid generalization. Too few patterns may cause "over-fitting" (wherein the network performs well on the training data set, but poorly on independent test patterns drawn from the same distribution as the training patterns, as in Figure A3).

Computational complexity refers to the time required for a learning algorithm to estimate a solution from training patterns. Many existing learning algorithms have high computational complexity. Designing efficient algorithms for neural network learning is a very active research topic.

There are four basic types of learning rules: error-correction, Boltzmann, Hebbian, and competitive learning.

ERROR-CORRECTION RULES. In the supervised learning paradigm, the network is given a desired output for each input pattern. During the learning process, the actual output y generated by the network may not equal the desired output d. The basic principle of error-correction learning rules is to use the error signal $(d-y)$ to modify the connection weights to gradually reduce this error.

The perceptron learning rule is based on this error-correction principle. A perceptron consists of a single neuron with adjustable weights, $w_j, j = 1, 2, \dots, n$, and threshold u, as shown in Figure 2. Given an input vector $\mathbf{x} = (x_1, x_2, \dots, x_n)^t$, the net input to the neuron is

Perceptron learning algorithm

1. Initialize the weights and threshold to small random numbers.
2. Present a pattern vector $(x_1, x_2, \ldots, x_n)^t$ and evaluate the output of the neuron.
3. Update the weights according to

   $$w_j(t+1) = w_j(t) + \eta\,(d - y)\,x_j,$$

   where d is the desired output, t is the iteration number, and η $(0.0 < \eta < 1.0)$ is the gain (step size).

$$v = \sum_{j=1}^{n} w_j x_j - u$$

The output y of the perceptron is $+1$ if $v > 0$, and 0 otherwise. In a two-class classification problem, the perceptron assigns an input pattern to one class if $y = 1$, and to the other class if $y=0$. The linear equation

$$\sum_{j=1}^{n} w_j x_j - u = 0$$

defines the decision boundary (a hyperplane in the n-dimensional input space) that halves the space.

Rosenblatt[5] developed a learning procedure to determine the weights and threshold in a perceptron, given a set of training patterns (see the "Perceptron learning algorithm" sidebar).

Note that learning occurs only when the perceptron makes an error. Rosenblatt proved that when training patterns are drawn from two linearly separable classes, the perceptron learning procedure converges after a finite number of iterations. This is the *perceptron convergence theorem*. In practice, you do not know whether the patterns are linearly separable. Many variations of this learning algorithm have been proposed in the literature.[2] Other activation functions that lead to different learning characteristics can also be used. However, *a single-layer perceptron can only separate linearly separable patterns as long as a monotonic activation function is used.*

The back-propagation learning algorithm (see the "Back-propagation algorithm sidebar") is also based on the error-correction principle.

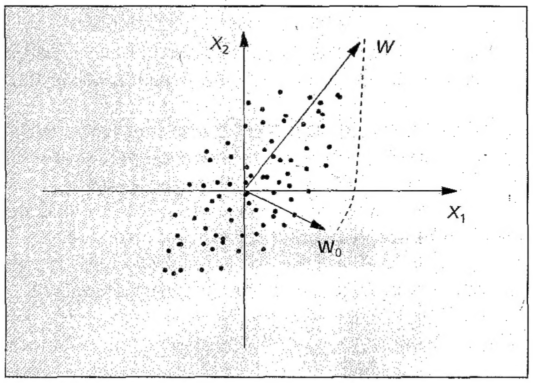


Figure 5. Orientation selectivity of a single neuron trained using the Hebbian rule.

Boltzmann Learning. Boltzmann machines are symmetric recurrent networks consisting of binary units (+1 for "on" and −1 for "off"). By symmetric, we mean that the weight on the connection from unit i to unit j is equal to the weight on the connection from unit j to unit i ($w_{ij} = w_{ji}$). A subset of the neurons, called *visible*, interact with the environment; the rest, called *hidden*, do not. Each neuron is a stochastic unit that generates an output (or state) according to the Boltzmann distribution of statistical mechanics. Boltzmann machines operate in two modes: *clamped*, in which visible neurons are clamped onto specific states determined by the environment; and *free-running*, in which both visible and hidden neurons are allowed to operate freely.

Boltzmann learning is a stochastic learning rule derived from information-theoretic and thermodynamic principles.[10] The objective of Boltzmann learning is to adjust the connection weights so that the states of visible units satisfy a particular desired probability distribution. According to the Boltzmann learning rule, the change in the connection weight w_{ij} is given by

$$\Delta w_{ij} = \eta(\bar{\rho}_{ij} - \rho_{ij}),$$

where η is the learning rate, and $\bar{\rho}_{ij}$ and ρ_{ij} are the correlations between the states of units i and j when the network operates in the clamped mode and free-running mode, respectively. The values of $\bar{\rho}_{ij}$ and ρ_{ij} are usually estimated from Monte Carlo experiments, which are extremely slow.

Boltzmann learning can be viewed as a special case of error-correction learning in which error is measured not as the direct difference between desired and actual outputs, but as the difference between the correlations among the outputs of two neurons under clamped and free-running operating conditions.

Hebbian Rule. The oldest learning rule is *Hebb's postulate of learning*.[13] Hebb based it on the following observation from neurobiological experiments: If neurons on both sides of a synapse are activated synchronously and repeatedly, the synapse's strength is selectively increased.

Mathematically, the Hebbian rule can be described as

$$w_{ij}(t+1) = w_{ij}(t) + \eta\, y_j(t)\, x_i(t),$$

where x_i and y_j are the output values of neurons i and j, respectively, which are connected by the synapse w_{ij}, and η is the learning rate. Note that x_i is the input to the synapse.

An important property of this rule is that learning is done locally, that is, the change in synapse weight depends only on the activities of the two neurons connected by it. This significantly simplifies the complexity of the learning circuit in a VLSI implementation.

A single neuron trained using the Hebbian rule exhibits an orientation selectivity. Figure 5 demonstrates this property. The points depicted are drawn from a two-dimensional Gaussian distribution and used for training a neuron. The weight vector of the neuron is initialized to $\mathbf{w}_0$ as shown in the figure. As the learning proceeds, the weight vector moves progressively closer to the direction $\mathbf{w}$ of maximal variance in the data. In fact, $\mathbf{w}$ is the eigenvector of the covariance matrix of the data corresponding to the largest eigenvalue.

COMPETITIVE LEARNING RULES. Unlike Hebbian learning (in which multiple output units can be fired simultaneously), competitive-learning output units compete among themselves for activation. As a result, only one output unit is active at any given time. This phenomenon is known as *winner-take-all*. Competitive learning has been found to exist in biological neural networks.[3]

Competitive learning often clusters or categorizes the input data. Similar patterns are grouped by the network and represented by a single unit. This grouping is done automatically based on data correlations.

The simplest competitive learning network consists of a single layer of output units as shown in Figure 4. Each output unit i in the network connects to all the input units (x_j's) via weights, $w_{ij}, j = 1, 2, \ldots, n$. Each output unit also connects to all other output units via inhibitory weights but has a self-feedback with an excitatory weight. As a result of competition, only the unit i^* with the largest (or the smallest) net input becomes the winner, that is, $\mathbf{w}_{i^*} \cdot \mathbf{x} \geq \mathbf{w}_i \cdot \mathbf{x}, \forall i$, or $\| \mathbf{w}_{i^*} - \mathbf{x} \| \leq \| \mathbf{w}_i - \mathbf{x} \|, \forall i$. When all the weight vectors are normalized, these two inequalities are equivalent.

A simple competitive learning rule can be stated as

$$\Delta w_{ij} = \begin{cases} \eta(x_j^u - w_{i^*j}), & i = i^*, \\ 0, & i \neq i^*. \end{cases} \qquad (1)$$

Note that only the weights of the winner unit get updated. The effect of this learning rule is to move the stored pattern in the winner unit (weights) a little bit closer to the input pattern. Figure 6 demonstrates a geometric interpretation of competitive learning. In this example, we assume that all input vectors have been normalized to have unit length. They are depicted as black dots in Figure 6. The weight vectors of the three units are randomly initialized. Their initial and final positions on the sphere after competitive learning are marked as Xs in Figures 6a and 6b, respectively. In Figure 6, each of the three natural groups (clusters) of patterns has been discovered by an output unit whose weight vector points to the center of gravity of the discovered group.

You can see from the competitive learning rule that the network will not stop learning (updating weights) unless the learning rate η is 0. A particular input pattern can fire different output units at different iterations during learning. This brings up the stability issue of a learning system. The system is said to be *stable* if no pattern in the training data changes its category after a finite number of learning iterations. One way to achieve stability is to force the learning rate to decrease gradually as the learning process proceeds towards 0. However, this artificial freezing of learning causes another problem termed *plasticity*, which is the ability to adapt to new data. This is known as Grossberg's *stability-plasticity* dilemma in competitive learning.

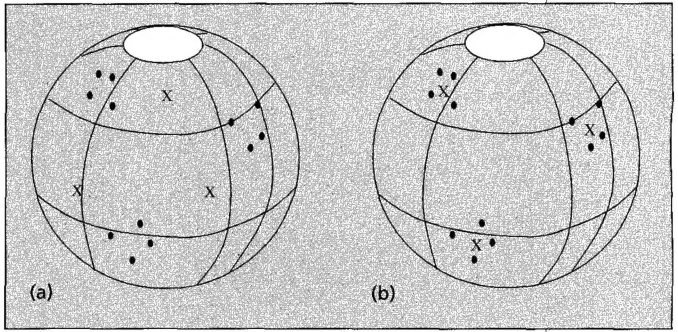


Figure 6. An example of competitive learning: (a) before learning; (b) after learning.

The most well-known example of competitive learning is *vector quantization* for data compression. It has been widely used in speech and image processing for efficient storage, transmission, and modeling. Its goal is to represent a set or distribution of input vectors with a relatively small number of prototype vectors (weight vectors), or a codebook. Once a codebook has been constructed and agreed upon by both the transmitter and the receiver, you need only transmit or store the index of the corresponding prototype to the input vector. Given an input vector, its corresponding prototype can be found by searching for the nearest prototype in the codebook.

SUMMARY. Table 2 summaries various learning algorithms and their associated network architectures (this is not an exhaustive list). Both supervised and unsupervised learning paradigms employ learning rules based

Back-propagation algorithm

1. Initialize the weights to small random values.
2. Randomly choose an input pattern $\mathbf{x}^{(\mu)}$.
3. Propagate the signal forward through the network.
4. Compute δ_i^L in the output layer ($o_i = y_i^L$)

   $$\delta_i^L = g'(h_i^L)\left[d_i^u - y_i^L\right],$$

   where h_i^l represents the net input to the ith unit in the lth layer, and g' is the derivative of the activation function g.
5. Compute the deltas for the preceding layers by propagating the errors backwards;

   $$\delta_i^l = g'(h_i^l)\sum_j w_{ij}^{l+1}\delta_j^{l+1},$$

   for $l = (L-1), \ldots, 1$.
6. Update weights using

   $$\Delta w_{ji}^l = \eta \delta_i^l y_j^{l-1}$$

7. Go to step 2 and repeat for the next pattern until the error in the output layer is below a prespecified threshold or a maximum number of iterations is reached.

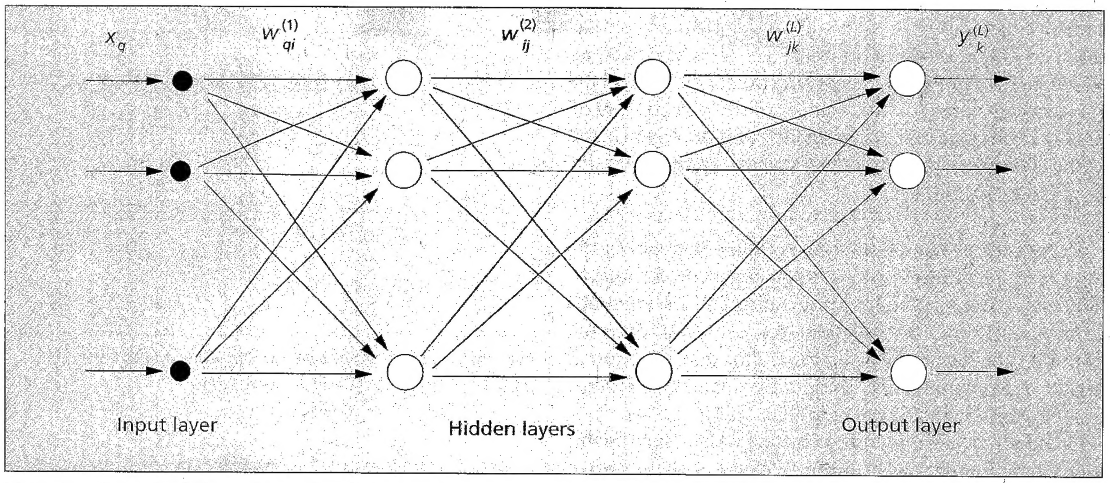


Figure 7. A typical three-layer feed-forward network architecture.

on error-correction, Hebbian, and competitive learning. Learning rules based on error-correction can be used for training feed-forward networks, while Hebbian learning rules have been used for all types of network architectures. However, each learning algorithm is designed for training a specific architecture. Therefore, when we discuss a learning algorithm, a particular network architecture association is implied. Each algorithm can

Table 2. Well-known learning algorithms.

Paradigm	Learning rule	Architecture	Learning algorithm	Task
Supervised	Error-correction	Single- or multilayer perceptron	Perceptron learning algorithms Back-propagation Adaline and Madaline	Pattern classification Function approximation Prediction, control
	Boltzmann	Recurrent	Boltzmann learning algorithm	Pattern classification
	Hebbian	Multilayer feed-forward	Linear discriminant analysis	Data analysis Pattern classification
	Competitive	Competitive	Learning vector quantization	Within-class categorization Data compression
		ART network	ARTMap	Pattern classification Within-class categorization
Unsupervised	Error-correction	Multilayer feed-forward	Sammon's projection	Data analysis
	Hebbian	Feed-forward or competitive	Principal component analysis	Data analysis Data compression
		Hopfield Network	Associative memory learning	Associative memory
	Competitive	Competitive	Vector quantization	Categorization Data compression
		Kohonen's SOM	Kohonen's SOM	Categorization Data analysis
		ART networks	ART1, ART2	Categorization
Hybrid	Error-correction and competitive	RBF network	RBF learning algorithm	Pattern classification Function approximation Prediction, control

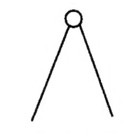
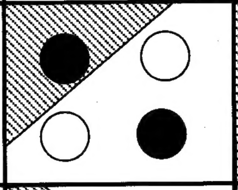
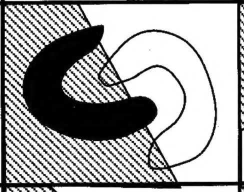
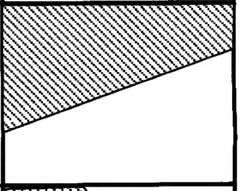
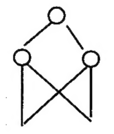
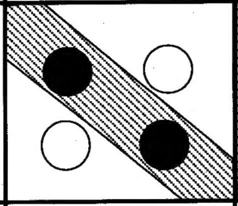
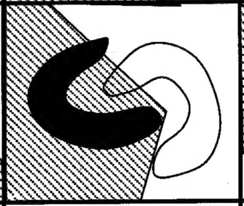
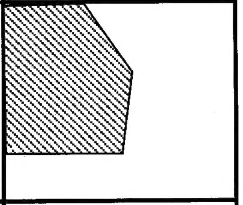
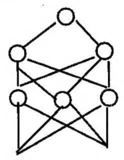
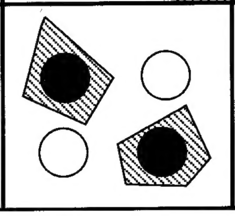
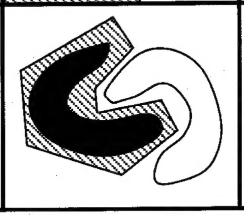
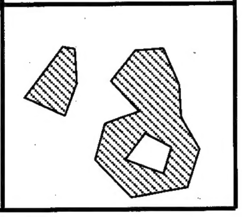
Structure	Description of decision regions	Exclusive-OR problem	Classes with meshed regions	General region shapes
 Single layer	Half plane bounded by hyperplane			
 Two layer	Arbitrary (complexity limited by number of hidden units)			
 Three layer	Arbitrary (complexity limited by number of hidden units)			

Figure 8. A geometric interpretation of the role of hidden unit in a two-dimensional input space.

perform only a few tasks well. The last column of Table 2 lists the tasks that each algorithm can perform. Due to space limitations, we do not discuss some other algorithms, including Adaline, Madaline,[14] linear discriminant analysis,[15] Sammon's projection,[15] and principal component analysis.[2] Interested readers can consult the corresponding references (this article does not always cite the first paper proposing the particular algorithms).

MULTILAYER FEED-FORWARD NETWORKS

Figure 7 shows a typical three-layer perceptron. In general, a standard L-layer feed-forward network (we adopt the convention that the input nodes are not counted as a layer) consists of an input stage, $(L-1)$ hidden layers, and an output layer of units successively connected (fully or locally) in a feed-forward fashion with no connections between units in the same layer and no feedback connections between layers.

Multilayer perceptron

The most popular class of multilayer feed-forward networks is *multilayer perceptrons* in which each computational unit employs either the thresholding function or the sigmoid function. Multilayer perceptrons can form arbitrarily complex decision boundaries and represent any Boolean function.[6] The development of the *back-propagation* learning algorithm for determining weights in a multilayer perceptron has made these networks the most popular among researchers and users of neural networks.

We denote $w_{ij}^{(l)}$ as the weight on the connection between the ith unit in layer $(l-1)$ to jth unit in layer l.

Let $\{(\mathbf{x}^{(1)}, \mathbf{d}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{d}^{(2)}), \ldots, (\mathbf{x}^{(p)}, \mathbf{d}^{(p)})\}$ be a set of p training patterns (input-output pairs), where $\mathbf{x}^{(i)} \in R^n$ is the input vector in the n-dimensional pattern space, and $\mathbf{d}^{(i)} \in [0, 1]^m$, an m-dimensional hypercube. For classification purposes, m is the number of classes. The squared-error cost function most frequently used in the ANN literature is defined as

$$E = \frac{1}{2}\sum_{i=1}^{p} \left\| \mathbf{y}^{(i)} - \mathbf{d}^{(i)} \right\|^2 \qquad (2)$$

The back-propagation algorithm[9] is a gradient-descent method to minimize the squared-error cost function in Equation 2 (see "Back-propagation algorithm" sidebar).

A geometric interpretation (adopted and modified from Lippmann[14]) shown in Figure 8 can help explicate the role of hidden units (with the threshold activation function).

Each unit in the first hidden layer forms a hyperplane in the pattern space; boundaries between pattern classes can be approximated by hyperplanes. A unit in the second hidden layer forms a hyperregion from the outputs of the first-layer units; a decision region is obtained by performing an AND operation on the hyperplanes. The output-layer units combine the decision regions made by the units in the second hidden layer by performing logical OR operations. Remember that this scenario is depicted only to explain the role of hidden units. Their actual behavior, after the network is trained, could differ.

A two-layer network can form more complex decision boundaries than those shown in Figure 8. Moreover, multilayer perceptrons with sigmoid activation functions can form smooth decision boundaries rather than piecewise linear boundaries.

Radial Basis Function network

The Radial Basis Function (RBF) network,[3] which has two layers, is a special class of multilayer feed-forward networks. Each unit in the hidden layer employs a radial basis function, such as a Gaussian kernel, as the activation function. The radial basis function (or kernel function) is centered at the point specified by the weight vector associated with the unit. Both the positions and the widths of these kernels must be learned from training patterns. There are usually many fewer kernels in the RBF network than there are training patterns. Each output unit implements a linear combination of these radial basis functions. From the point of view of function approximation, the hidden units provide a set of functions that constitute a basis set for representing input patterns in the space spanned by the hidden units.

There are a variety of learning algorithms for the RBF network.[3] The basic one employs a two-step learning strategy, or hybrid learning. It estimates kernel positions and kernel widths using an unsupervised clustering algorithm, followed by a supervised least mean square (LMS) algorithm to determine the connection weights between the hidden layer and the output layer. Because the output units are linear, a noniterative algorithm can be used. After this initial solution is obtained, a supervised gradient-based algorithm can be used to refine the network parameters.

This hybrid learning algorithm for training the RBF network converges much faster than the back-propagation algorithm for training multilayer perceptrons. However, for many problems, the RBF network often involves a larger number of hidden units. This implies that the runtime (after training) speed of the RBF network is often slower than the runtime speed of a multilayer perceptron. The efficiencies (error versus network size) of the RBF network and the multilayer perceptron are, however, problem-dependent. It has been shown that the RBF network has the same asymptotic approximation power as a multilayer perceptron.

SOM learning algorithm

1. Initialize weights to small random numbers; set initial learning rate and neighborhood.
2. Present a pattern **x**, and evaluate the network outputs.
3. Select the unit (c_i, c_j) with the minimum output:

$$\|\mathbf{x} - \mathbf{w}_{c_i c_j}\| = \min_{ij} \|\mathbf{x} - \mathbf{w}_{ij}\|$$

4. Update all weights according to the following learning rule:

$$\mathbf{w}_{ij}(t+1) = \begin{cases} \mathbf{w}_{ij}(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{w}_{ij}(t)], & \text{if } (i,j) \in N_{c_i c_j}(t), \\ \mathbf{w}_{ij}(t), & \text{otherwise,} \end{cases}$$

   where $N_{c_i c_j}(t)$ is the neighborhood of the unit (c_i, c_j) at time t, and $\alpha(t)$ is the learning rate.
5. Decrease the value of $\alpha(t)$ and shrink the neighborhood $N_{c_i c_j}(t)$.
6. Repeat steps 2 through 5 until the change in weight values is less than a prespecified threshold or a maximum number of iterations is reached.

Issues

There are many issues in designing feed-forward networks, including

- how many layers are needed for a given task,
- how many units are needed per layer,
- how will the network perform on data not included in the training set (generalization ability), and
- how large the training set should be for "good" generalization.

Although multilayer feed-forward networks using back-propagation have been widely employed for classification and function approximation,[2] many design parameters still must be determined by trial and error. Existing theoretical results provide only very loose guidelines for selecting these parameters in practice.

KOHONEN'S SELF-ORGANIZING MAPS

The self-organizing map (SOM)[16] has the desirable property of topology preservation, which captures an important aspect of the feature maps in the cortex of highly developed animal brains. In a topology-preserving mapping, nearby input patterns should activate nearby output units on the map. Figure 4 shows the basic network architecture of Kohonen's SOM. It basically consists of a two-dimensional array of units, each connected to all n input nodes. Let $\mathbf{w}_{ij}$ denote the n-dimensional vector associated with the unit at location (i, j) of the 2D array. Each neuron computes the Euclidean distance between the input vector $\mathbf{x}$ and the stored weight vector $\mathbf{w}_{ij}$.

This SOM is a special type of competitive learning network that defines a spatial neighborhood for each output unit. The shape of the local neighborhood can be square, rectangular, or circular. Initial neighborhood size is often set to one half to two thirds of the network size and shrinks over time according to a schedule (for example, an exponentially decreasing function). During competitive learning, all the weight vectors associated with the winner and its neighboring units are updated (see the "SOM learning algorithm" sidebar).

Kohonen's SOM can be used for projection of multivariate data, density approximation, and clustering. It has been successfully applied in the areas of speech recognition, image processing, robotics, and process control.[2] The design parameters include the dimensionality of the neuron array, the number of neurons in each dimension, the shape of the neighborhood, the shrinking schedule of the neighborhood, and the learning rate.

ADAPTIVE RESONANCE THEORY MODELS

Recall that the *stability-plasticity* dilemma is an important issue in competitive learning. How do we learn new things (plasticity) and yet retain the stability to ensure that existing knowledge is not erased or corrupted? Carpenter and Grossberg's Adaptive Resonance Theory models (ART1, ART2, and ARTMap) were developed in an attempt to overcome this dilemma.[17] The network has a sufficient supply of output units, but they are not used until deemed necessary. A unit is said to be *committed* (*uncommitted*) if it is (is not) being used. The learning algorithm updates the stored prototypes of a category only if the input vector is sufficiently similar to them. An input vector and a stored prototype are said to resonate when they are sufficiently similar. The extent of similarity is controlled by a *vigilance parameter*, ρ, with $0 < \rho < 1$, which also determines the number of categories. When the input vector is not sufficiently similar to any existing prototype in the network, a new category is created, and an uncommitted unit is assigned to it with the input vector as the initial prototype. If no such uncommitted unit exists, a novel input generates no response.

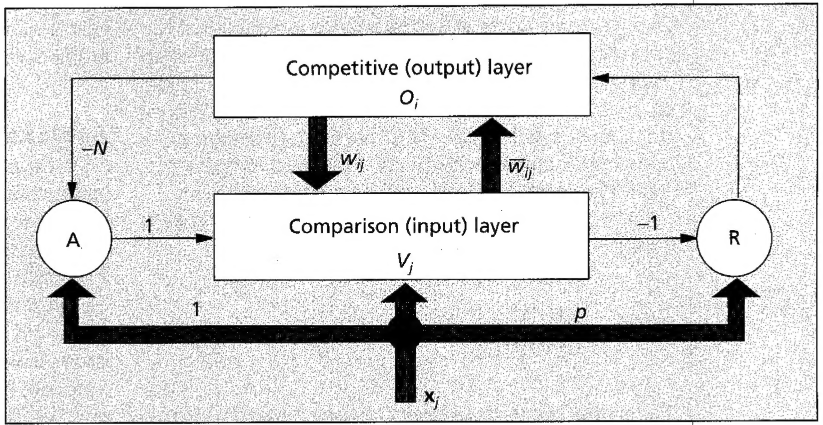


Figure 9. ART1 network.

We present only ART1, which takes binary (0/1) input to illustrate the model. Figure 9 shows a simplified diagram of the ART1 architecture.[2] It consists of two layers of fully connected units. A top-down weight vector $\mathbf{w}_j$ is associated with unit j in the input layer, and a bottom-up weight vector $\overline{\mathbf{w}}_i$ is associated with output unit i; $\overline{\mathbf{w}}_i$ is the normalized version of $\mathbf{w}_i$.

$$\overline{\mathbf{w}}_i = \frac{\mathbf{w}_i}{\varepsilon + \sum_j w_{ji}}, \qquad (3)$$

where ε is a small number used to break the ties in selecting the winner. The top-down weight vectors $\mathbf{w}_j$'s store cluster prototypes. The role of normalization is to prevent prototypes with a long vector length from dominating prototypes with a short one. Given an n-bit input vector $\mathbf{x}$, the output of the auxiliary unit A is given by

$$A = Sgn_{0/1}\left(\sum_j x_j - n\sum_i O_i - 0.5\right),$$

where $Sgn_{0/1}(x)$ is the *signum* function that produces +1 if $x \geq 0$ and 0 otherwise, and the output of an input unit is given by

$$V_j = Sgn_{0/1}\left(x_j + \sum_i w_{ji}O_i + A - 1.5\right)$$
$$= \begin{cases} x_j, & \text{if no output } O_j \text{ is "on"}, \\ x_j \wedge \Sigma_i w_{ji} O_i, & \text{otherwise.} \end{cases}$$

A reset signal R is generated only when the similarity is less than the vigilance level. (See the "ART1 learning algorithm" sidebar.)

The ART1 model can create new categories and reject an input pattern when the network reaches its capacity. However, the number of categories discovered in the input data by ART1 is sensitive to the vigilance parameter.

HOPFIELD NETWORK

Hopfield used a network *energy* function as a tool for designing recurrent networks and for understanding their dynamic behavior.[7] Hopfield's formulation made explicit the principle of storing information as dynamically stable attractors and popularized the use of recurrent networks for associative memory and for solving combinatorial optimization problems.

A Hopfield network with n units has two versions: binary and continuously valued. Let v_i be the state or output of the ith unit. For binary networks, v_i is either +1 or −1, but for continuous networks, v_i can be any value between 0 and 1. Let w_{ij} be the synapse weight on the connection from units i to j. In Hopfield networks, $w_{ij} = w_{ji}$, $\forall i, j$ (symmetric networks), and $w_{ii} = 0$, $\forall i$ (no self-feedback connections). The network dynamics for the binary Hopfield network are

$$v_i = Sgn\left(\sum_j w_{ij}v_j - \theta_i\right) \qquad (4)$$

ART1 learning algorithm

1. Initialize $w_{ij} = 1$, for all i, j. Enable all the output units.
2. Present a new pattern $\mathbf{x}$.
3. Find the winner unit i^* among the enabled output units

   $$\overline{\mathbf{w}}_{i^*} \cdot \mathbf{x} \geq \overline{\mathbf{w}}_i \cdot \mathbf{x}, \forall i$$

4. Perform vigilance test

   $$r = \frac{\mathbf{w}_{i^*} \cdot \mathbf{x}}{\Sigma_j x_j}$$

   If $r \geq \rho$ (resonance), go to step 5. Otherwise, disable unit i^* and go to step 3 (until all the output units are disabled).
5. Update the winning weight vector $\mathbf{w}_{i^*}$, enable all the output units, and go to step 2

   $$\Delta w_{ji^*} = \eta\,(V_j - w_{ji^*})$$

6. If all output units are disabled, select one of the uncommitted output units and set its weight vector to x. If there is no uncommitted output unit (capacity is reached), the network rejects the input pattern.

The dynamic update of network states in Equation 4 can be carried out in at least two ways: *synchronously* and *asynchronously*. In a synchronous updating scheme, all units are updated simultaneously at each time step. A central clock must synchronize the process. An asynchronous updating scheme selects one unit at a time and updates its state. The unit for updating can be randomly chosen.

The energy function of the binary Hopfield network in a state $\mathbf{v} = (v_1, v_2, \ldots, v_n)^T$ is given by

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} v_i v_j \qquad (5)$$

The central property of the energy function is that as network state evolves according to the network dynamics (Equation 4), the network energy always decreases and eventually reaches a local minimum point (attractor) where the network stays with a constant energy.

Associative memory

When a set of patterns is stored in these network attractors, it can be used as an *associative memory*. Any pattern present in the basin of attraction of a stored pattern can be used as an index to retrieve it.

An associative memory usually operates in two phases: storage and retrieval. In the storage phase, the weights in the network are determined so that the attractors of the network memorize a set of p n-dimensional patterns $\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^p\}$ to be stored. A generalization of the Hebbian learning rule can be used for setting connection weights w_{ij}. In the retrieval phase, the input pattern is used as the initial state of the network, and the network evolves according to its dynamics. A pattern is produced (or retrieved) when the network reaches equilibrium.

How many patterns can be stored in a network with n binary units? In other words, what is the memory capacity of a network? It is finite because a network with n binary units has a maximum of 2^n distinct states, and not all of them are attractors. Moreover, not all attractors (stable states) can store useful patterns. Spurious attractors can also store patterns different from those in the training set.[2]

It has been shown that the maximum number of random patterns that a Hopfield network can store is $P_{\max} \approx 0.15n$. When the number of stored patterns $p < 0.15n$, a nearly perfect recall can be achieved. When memory patterns are orthogonal vectors instead of random patterns, more patterns can be stored. But the number of spurious attractors increases as p reaches capacity. Several learning rules have been proposed for increasing the memory capacity of Hopfield networks.[2] Note that we require n^2 connections in the network to store p n-bit patterns.

Energy minimization

Hopfield networks always evolve in the direction that leads to lower network energy. This implies that if a combinatorial optimization problem can be formulated as minimizing this energy, the Hopfield network can be used to find the optimal (or suboptimal) solution by letting the network evolve freely. In fact, any quadratic objective function can be rewritten in the form of Hopfield network energy. For example, the classic Traveling Salesman Problem can be formulated as such a problem.

APPLICATIONS

We have discussed a number of important ANN models and learning algorithms proposed in the literature. They have been widely used for solving the seven classes of problems described in the beginning of this article. Table 2 showed typical suitable tasks for ANN models and learning algorithms. Remember that to successfully work with real-world problems, you must deal with numerous design issues, including network model, network size, activation function, learning parameters, and number of training samples. We next discuss an optical character recognition (OCR) application to illustrate how multilayer feed-forward networks are successfully used in practice.

OCR deals with the problem of processing a scanned image of text and transcribing it into machine-readable form. We outline the basic components of OCR and explain how ANNs are used for character classification.

An OCR system

An OCR system usually consists of modules for preprocessing, segmentation, feature extraction, classification, and contextual processing. A paper document is scanned to produce a gray-level or binary (black-and-white) image. In the preprocessing stage, filtering is applied to remove noise, and text areas are located and converted to a binary image using a global or local adaptive thresholding method. In the segmentation step, the text image is separated into individual characters. This is a particularly difficult task with handwritten text, which contains a proliferation of touching characters. One effective technique is to break the composite pattern into smaller patterns (over-segmentation) and find the correct character segmentation points using the output of a pattern classifier.

Because of various degrees of slant, skew, and noise level, and various writing styles, recognizing segmented characters is not easy. This is evident from Figure 10, which shows the size-normalized character bitmaps of a sample set from the NIST (National Institute of Standards and Technology) hand-print character database.[18]

Schemes

Figure 11 shows the two main schemes for using ANNs in an OCR system. The first one employs an explicit feature extractor (not necessarily a neural network). For instance, contour direction features are used in Figure 11. The extracted features are passed to the input stage of a multilayer feed-forward network.[19] This scheme is very flexible in incorporating a large variety of features. The other scheme does not explicitly extract features from the raw data. The feature extraction implicitly takes place within the intermediate stages (hidden layers) of the ANN. A nice property of this scheme is that feature extraction and classification are integrated and trained simultaneously to produce optimal classification results. It is not clear whether the types of features that can be extracted by this integrated architecture are the most effective for character recognition. Moreover, this scheme requires a much larger network than the first one.

A typical example of this integrated feature extraction-classification scheme is the network developed by Le Cun et al.[20] for zip code recognition. A 16 × 16 normalized gray-level image is presented to a feed-forward network with three hidden layers. The units in the first layer are locally connected to the units in the input layer, forming a set of local feature maps. The second hidden layer is constructed in a similar way. Each unit in the second layer also combines local information coming from feature maps in the first layer.

The activation level of an output unit can be interpreted as an approximation of the a posteriori probability of the input pattern's belonging to a particular class. The output categories are ordered according to activation levels and passed to the post-processing stage. In this stage, contextual information is exploited to update the classifier's output. This could, for example, involve looking up a dictionary of admissible words, or utilizing syntactic constraints present, for example, in phone or social security numbers.

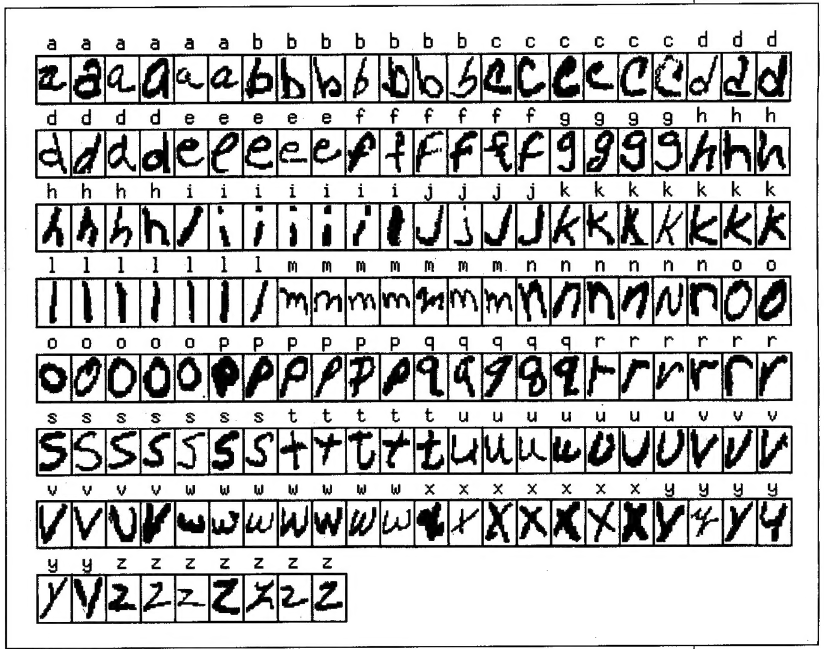


Figure 10. A sample set of characters in the NIST database.

Results

ANNs work very well in the OCR application. However, there is no conclusive evidence about their superiority over conventional statistical pattern classifiers. At the First Census Optical Character Recognition System Conference held in 1992,[18] more than 40 different handwritten character recognition systems were evaluated based on their performance on a common database. The top 10 performers used either some type of multilayer feed-forward network or a nearest neighbor-based classifier. ANNs tend to be superior in terms of speed and memory requirements compared to nearest neighbor methods. Unlike the nearest neighbor methods, classification speed using ANNs is independent of the size of the training set. The recognition accuracies of the top OCR systems on the NIST isolated (presegmented) character data were above 98 percent for digits, 96 percent for uppercase characters, and 87 percent for lowercase characters. (Low recognition accuracy for lowercase characters was largely due to the fact that the test data differed significantly from the training data, as well as being due to "ground-truth" errors.) One conclusion drawn from the test is that OCR system performance on isolated characters compares well with human performance. However, humans still outperform OCR systems on unconstrained and cursive handwritten documents.

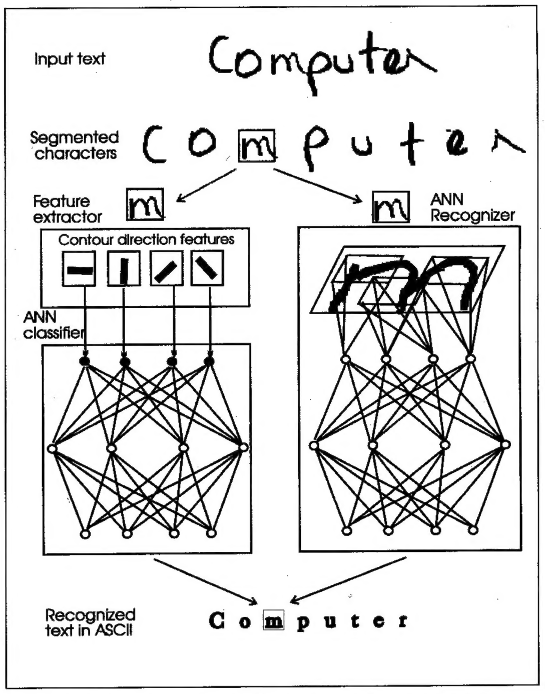


Figure 11. Two schemes for using ANNs in an OCR system.

DEVELOPMENTS IN ANNS HAVE STIMULATED a lot of enthusiasm and criticism. Some comparative studies are optimistic, some offer pessimism. For many tasks, such as pattern recognition, no one approach dominates the others. The choice of the best technique should be driven by the given application's nature. We should try to understand the capacities, assumptions, and applicability of various approaches and maximally exploit their complementary advantages to develop better intelligent systems. Such an effort may lead to a synergistic approach that combines the strengths of ANNs with other technologies to achieve significantly better performance for challenging problems. As Minsky[21] recently observed, the time has come to build systems out of diverse components. Individual modules are important, but we also need a good methodology for integration. It is clear that communication and cooperative work between researchers working in ANNs and other disciplines will not only avoid repetitious work but (and more important) will stimulate and benefit individual disciplines. ∎

Acknowledgments

We thank Richard Casey (IBM Almaden); Pat Flynn (Washington State University); William Punch, Chitra Dorai, and Kalle Karu (Michigan State University); Ali Khotanzad (Southern Methodist University); and Ishwar Sethi (Wayne State University) for their many useful suggestions.

References

1. *DARPA Neural Network Study*, AFCEA Int'l Press, Fairfax, Va., 1988.
2. J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Reading, Mass., 1991.
3. S. Haykin, *Neural Networks: A Comprehensive Foundation*, MacMillan College Publishing Co., New York, 1994.
4. W.S. McCulloch and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bull. Mathematical Biophysics*, Vol. 5, 1943, pp. 115-133.
5. R. Rosenblatt, *Principles of Neurodynamics*, Spartan Books, New York, 1962.
6. M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, Mass., 1969.
7. J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," in *Proc. Nat'l Academy of Sciences*, USA 79, 1982, pp. 2,554-2,558.
8. P. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," PhD thesis, Dept. of Applied Mathematics, Harvard University, Cambridge, Mass., 1974.
9. D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, MIT Press, Cambridge, Mass., 1986.
10. J.A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, Mass., 1988.
11. S. Brunak and B. Lautrup, *Neural Networks, Computers with Intuition*, World Scientific, Singapore, 1990.
12. J. Feldman, M.A. Fanty, and N.H. Goddard, "Computing with Structured Neural Networks," *Computer*, Vol. 21, No. 3, Mar. 1988, pp. 91-103.
13. D.O. Hebb, *The Organization of Behavior*, John Wiley & Sons, New York, 1949.
14. R.P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, Vol. 4, No. 2, Apr. 1987, pp. 4-22.
15. A.K. Jain and J. Mao, "Neural Networks and Pattern Recognition," in *Computational Intelligence: Imitating Life*, J.M. Zurada, R. J. Marks II, and C.J. Robinson, eds., IEEE Press, Piscataway, N.J., 1994, pp. 194-212.
16. T. Kohonen, *Self Organization and Associative Memory*, Third Edition, Springer-Verlag, New York, 1989.
17. G.A. Carpenter and S. Grossberg, *Pattern Recognition by Self-Organizing Neural Networks*, MIT Press, Cambridge, Mass., 1991.
18. "The First Census Optical Character Recognition System Conference," R.A. Wilkinson et al., eds., . Tech. Report, NISTIR 4912, US Dept. Commerce, NIST, Gaithersburg, Md., 1992.
19. K. Mohiuddin and J. Mao, "A Comparative Study of Different Classifiers for Handprinted Character Recognition," in *Pattern Recognition in Practice IV*, E.S. Gelsema and L.N. Kanal, eds., Elsevier Science, The Netherlands, 1994, pp. 437-448.
20. Y. Le Cun et al., "Back-Propagation Applied to Handwritten Zip Code Recognition, *Neural Computation*, Vol. 1, 1989, pp. 541–551.
21. M. Minsky, "Logical Versus Analogical or Symbolic Versus Connectionist or Neat Versus Scruffy," *AI Magazine*, Vol. 65, No. 2, 1991, pp. 34-51.

Anil K. Jain *is a University Distinguished Professor and the chair of the Department of Computer Science at Michigan State University. His interests include statistical pattern recognition, exploratory pattern analysis, neural networks, Markov random fields, texture analysis, remote sensing, interpretation of range images, and 3D object recognition.*

Jain served as editor-in-chief of IEEE Transactions on Pattern Analysis and Machine Intelligence *from 1991 to 1994, and currently serves on the editorial boards of* Pattern Recognition, Pattern Recognition Letters, Journal of Mathematical Imaging, Journal of Applied Intelligence, *and* IEEE Transactions on Neural Networks. *He has coauthored, edited, and coedited numerous books in the field. Jain is a fellow of the IEEE and a speaker in the IEEE Computer Society's Distinguished Visitors Program for the Asia-Pacific region. He is a member of the IEEE Computer Society.*

Jianchang Mao *is a research staff member at the IBM Almaden Research Center. His interests include pattern recognition, neural networks, document image analysis, image processing, computer vision, and parallel computing.*

Mao received the BS degree in physics in 1983 and the MS degree in electrical engineering in 1986 from East China Normal University in Shanghai. He received the PhD in computer science from Michigan State University in 1994. Mao is the abstracts editor of IEEE Transactions on Neural Networks. *He is a member of the IEEE and the IEEE Computer Society.*

K.M. Mohiuddin *is the manager of the Document Image Analysis and Recognition project in the Computer Science Department at the IBM Almaden Research Center. He has led IBM projects on high-speed reconfigurable machines for industrial machine vision, parallel processing for scientific computing, and document imaging systems. His interests include document image analysis, handwriting recognition/OCR, data compression, and computer architecture.*

Mohiuddin received the MS and PhD degrees in electrical engineering from Stanford University in 1977 and 1982, respectively. He is an associate editor of IEEE Transactions on Pattern Analysis and Machine Intelligence. *He served on Computer's editorial board from 1984 to 1989, and is a senior member of the IEEE and a member of the IEEE Computer Society.*

Readers can contact Anil Jain at the Department of Computer Science, Michigan State University, A714 Wells Hall, East Lansing, MI 48824; jain@cps.msu.edu.